Examining the suitability of the Pattern Oriented Paradigm in the Educational Software Development Process

Alejandro Hossian, Enrique Sierra, Jorgelina Plaza, Adriana Acuña, Eugenia Luque, Ramón García Martínez

Facultad de Ingeniería
Universidad Nacional del Comahue
Buenos Aires 1400 – (8300) Neuquén
República Argentina
Contact: enriquesie@yahoo.com.ar, hossi@ietband.com.ar
Paper received on 04/08/08, accepted on 10/09/08.

Abstract. This paper examines the fitness of the Pattern-Oriented Paradigm, applied by Christopher Alexander to architectural systems, in the design of educational software products. As a result, the validity of replacing the traditional software development process by a new process started by finding the appropriate pattern that best addresses the educational product design problem, is discussed. An entire set of educational software development patterns is proposed and concluding remarks evaluate advantages and disadvantages that may arise as a consequence of the application of this new paradigm.

1 Patterns versus Methods: A new way to deal with complexity?

There are many problems that humans can deal with. As an example, a group of scientists and engineers has the challenge to design a hybrid energy supply system (a system that is made up of conventional and renewable energy sources) for a certain city. Even if the problem can be fairly complex, experts know the steps that they should follow to solve it. First of all, they have to predict (or at least estimate accurately) an energy demand profile for the city and also energy supply profiles for the sources, for a long time (could be ten or more years) into the future. This is not an easy task to do. However, there are some mathematical and computational tools that can be useful for this purpose: neural networks, ARMA models, etc. As a second step, the experts should determine the dimensions of the elements involved in the future installation. At this stage, the problem is almost solved, at least in its general framework. There might be some future details to adjust or define, but they are completely tractable.

In another design problem, a group of architects is required to construct an "environmental building". The building has not only to be integrated into its environment by exhibiting low energy 3consumption parameters (high levels of solar light, natural ventilation), but also has to be aesthetically integrated into the architectural styles

© M. G. Medina Barrera, J. F. Ramirez Cruz, J.H. Sossa Azuela, (Eds.).
Advances in Intelligent and Information Technologies.
Research in Computing Science 38, 2008, pp.343-353.

predominant in the city where the building is going to be constructed. Besides, most of its functions (security systems, anti-fire activation, heat and air conditional systems, etc.) are supposed to be automated, so that the building shows certain degree of "intelligence". But there is another constraint: the owners have to approve the overall project. It is quite obvious that it would be difficult to find a systematic method that could be applied to find, at least, a starting solution to this problem. There are so many variables to be considered, so many approaches to deal with the requirements, that the solution process is almost impossible to standardize. Or it can be affirmed that there is no solution process at all in the sense that the solution cannot be accomplished through a sequence of predefined steps. There is no systematic procedure to search for a solution. It is evident that in this case the group of architects in charge of the project will have to capitalize on their knowledge and experience to afford this challenge. Consciously or not, the experts will start looking for models or "patterns" that could best fit this new project. According to Christopher Alexander [1], "each pattern describes a problem which occurs over and over again in our environment, and then describes the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice".

The problems described above have elements in common: they are complex, and do not have a unique or optimal solution. However, the difference resides in the approach that must be pplied to solve them. Patterns seem to be a useful resource when there is no method or established process to attain the solution. Under certain circumstances, certain solutions apply, but there is no formal way to reach a prevailing one. Patterns seem to succeed where all methods fail.

2 The essentiality of human factor

A deeper analysis of the problem outlined in the previous section, leads to verify that the second one contains much more involvement with human aspects. As a design closer to the complexity of human life, it can be affirmed that it contains a larger extent of "human factor". According to certain authors, this is a typical feature of pattern-driven problems. Richard Gabriel [2] explains that "the pattern has a significant human component...all software serves human comfort or qualify of life; the best patterns explicitly appeal to aesthetics and utility". As Richard Gabriel points out, this degree of involvement with human life is essential to identify the convenience of applying a pattern or a combination of them to cope with a complex design problem. Furthermore, there is a high degree of human intervention in finding the most appropriate pattern themselves. Christopher Alexander affirms [1] that "...people should design for themselves their own houses, streets and communities. This idea may be radical (it implies a radical transformation of the architectural profession) but it comes simply from the observation that most of the wonderful places in the world were not made by architects but by people..."

A simple observation of the problem of software design shows that it shares most of the features of pattern-drives problems: high complexity, large extent of involvement with aspects of human life and so on. Moreover, educational software

exhibits a higher degree of involvement with human reality than any other type of software product. This fact encourages the development or finding of a set of patterns for learning environments.

3 Moving towards a user-centered approach

As discussed in the previous section, the human factor involved in any pattern-driven approach, arises the strategy of involving people as "users" or "participants" in the educational software design problem as an important issue to be borne in mind.

Users are a diverse bunch, and not just in computer skills. Each of us has a different (and very personal) way of approaching the world. And unlike computer skills, these differences are unlikely to change. People tend to have unique approaches to learning, for example. Some people prefer to absorb information visually, by way of maps, icons, movies or other images. For many others, information is most easily understood by reading or listening. Still other people have what is called a kinesthetic approach: they tend to learn by touching, trying, experimenting and charging ahead. Most of us use all these approaches at one time or another, depending on the task we are facing but tend to favor one approach over others [3].

Therefore, finding the right pattern which best fits certain learning environment implies performing a research process, as close to the user as possible, to find out the proper requirements and characteristics of the problem to be addressed by the design in a real context. Besides, different ways of involvement of the user in the whole process are highly recommended [4].

4 Finding the right patterns: eliciting educational software requirements

As Christopher Alexander [5] explains:"...Each pattern is a three part rule, which expresses a relation between a certain context, a problem and a solution. As an element in the world, each pattern is a relationship between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain spatial configuration which allow these forces to resolve themselves..."

It is quite clear from these words that a pattern can be described as a network of three interrelated elements: problem, context, and solution.

This fact implies that finding an adequate pattern to a given problem largely depends on a proper identification of the problem itself and the context in which it occurs. This phase of identification normally focuses on gathering information about the requirements of the system, along which many questions may arise such as: What is necessary to know about the project goals and mission? Who are going to be the users and what do they want? What sort of resources are available for the design? In which context will the learning activity take place?

By applying this research process at individual and group level, the educational software developing patterns exhibited on Tables 1, 2 and 3 have been identified [6][7]. As stated, fully understanding the problem and the context in which it takes place helps the designers to identify which pattern to apply. Normally, the problem

of finding the pattern or combination of patterns that best fits within the educational situation to be addressed implies a process of research and definition of software product requirements. If after this analysis, it can be concluded that the pattern to apply is still unknown (or has never been used before), a new pattern that is applicable for the project in development has to be created.

Table 1 Educational Software Development Patterns for Individual Learning Opportunities

Individual Learning Opportunities	Characteristics	Technology Impact
Tutorial	• Traditional • Linear • Reactive • Student is passive • Good for process training where adherence to process is required • Builds basic skills but not typically	• More Efficient • Self paced • Easily distributed • Can result in losing personal touch Example: Computer-based training of basic skills
Research-Construct	• research based • independently driven • user creates a tangible artifact	Example: http://www.scenemaker.com
Coaching	• JIT (Just in time) • Applied only when help is needed or requested • Otherwise similar to tutorial	• Provide quicker access to large amounts of material • Hyperlinks feasible • Always available Example: Help System
Simulation	• Useful in situation where complex, interacting events obscure cause and effect • Non threatening – can make mistakes without one notic-	• Requires technology; intensive use of technology Example: SimCity

	ing	
Current Information	• Keep apprised of things that are new • Used regularly, typically scanned • Should be annotated	• Allows for aggregation of information that is impractical in paper form • Immediate • Labor intensive. Requires an editorial function Example: My Yahoo
Meandering	• People like to wander and need tools to help reorient themselves • Historical tracking of what has been done useful at later points – should be able to save it • Might be valuable for teachers/researchers	• Efficient –users don't have to do anything as it can happen automatically • Difficult is determining what level of detail to capture and how to let users review that and how much to store. Example: Web browsers and back buttons
Scientific Method	• A process used to create experiments	• Can help structure, report and share information

Table 2 Educational Software Development Patterns for Small Group Opportunities

Small Group Opportunities	Characteristics	Technology Impact
Discussion Board	• Asynchronous • Captures history • Always available	• Searchable • Facilitates capture of history • Easy, practical entry of data • Usability is an issue

		Example: BBS
Chatting	• Synchronous • Captures history • Always available	• Brings together par ties of interest • Can be an enjoyable waste of time Example: Chat room
Expert View	• Provides structure • Facilitates entry to an unfamiliar setting • Uses stories	• Provide leverage • Incorporate multiple experts • Have to locate ex pert and can be ex pensive Example: Expert avail able through electronic medium
Annotations (Multiple Perspectives)	• Post-it notes • Connects directly to content of interest	• Searchable • Can be archived • Can have usability issues Example: insertion of a note in a Word Document
Peer to Peer Informal	• Allows people to brainstorm • Provides opportunity to transmit tacit knowledge • Opportunities for mentoring • Leads to mastery, subtly	• Difficult to capture the nonverbal aspects of a person's expertise through technology • Technology can at best facilitate this process or enhance it. Example: Building a paper airplane

5. Finding new Patterns

Once the designer has realized that he or she has to deal with a problem for which the pattern to apply is unknown or some aspects relevant to the solution remain obscure, there is no reason to fall into despair. A good analysis of require-

ments, that must be already done, is still useful. There are many learning theories at hand that can provide the designer with a set of instructional strategies depending upon the gathered requirements.

Table 3 Educational Software Development Patterns for Large Groups Learning Opportunities

Large Group Learning Opportunities	Characteristics	Technology Impact
Best Practices	• Expert proven process • Often needs to be tailored • Can overwhelm	• Makes systems usable • Effective information location • Easy Administration • Expensive, requires culture change Example: Electronic best practice system
Intelligent Agents	• Personalized Information • Filtering • Predictive recommendation ability • Adaptability	• Not feasible without technology • Expensive to install Example: Amazon.com, Firefly
Yellow pages	• Helps people find people • Show where knowledge resides	• Easy to find • Relevant expert Example: Electronic Expert Catalog

There are many learning theories to support instructional design, although the three main ones will be briefly discussed in this paper: behaviorism, cognitivism and constructivism.

"Behavioral learning theory is restricted to external, observable behaviors attempting to explain *why behaviors occur*. It is based on the premise that learning re-

sults from the pairing of responses to stimuli. Reinforcement is viewed as an event that follows the response. Behaviorism is based on four principles: *contiguity* (the response should follow the stimulus without delay), *repetition* (practices strength learning and improves retention), *feedback and reinforcement* (knowledge concerning the correctness of the response contributes to learning), *prompting and fading* (leading to the desired response under decreasingly cued conditions). By observing the four principles discussed, teachers and trainers have produced increasingly efficient instruction..." [8].

According to behaviorism, the first step is to specify the behavioral outcomes desired from the instruction. A "task analysis" is done in order to derive these behavioral outcomes or objectives [9]. The required instructional strategies are defined by this set of desired behaviors through the application of the four principles of the instruction listed above.

"In contrast to behavioral theory, cognitive theory attempts to determine *how learning takes places*, based on the process believed to occur within the learner..." [8]. Thus, taking a Cognitive Science approach (as opposed to behavioral approach), it is necessary to carry out a "cognitive task analysis" in order to completely understand what knowledge in what forms (representations) is needed to do a task and thus is needed as the outcomes of the instruction in how to do the task. It has been found, [10], four different kinds of knowledge to be the most useful in analyzing the knowledge involved in a variety of topics and tasks: factual knowledge, image knowledge, procedural knowledge and mental models. Instructional strategies (inference, interpretation, proceduralization, envision and simulation) arise as a result of the type of knowledge involved in the topic to be instructed and the cognitive task analysis previously developed.

Constructivism is a philosophy of learning founded on the premise that knowledge does not exist external to the learner (as opposed to behaviorism and cognitivism), but *it is internally constructed through a process of reflection on the learner's own experiences* [8]. Under constructivism, the learner constructs his own meaning of the world he lives in. According to this theory, the context of learning plays a very important role in the process of internal knowledge construction. Instructional strategies consist of providing the learner with authentic, meaningful experiences in relevant contexts that allow him to transfer much of the acquired knowledge or skills to real-world environments.

Learning theories are partial visions in the sense that, in general they contemplate certain aspects of the overall learning situation [8]. Depending on the characteristics of the learning environment to be designed, it is the developer's duty to determine the right mix of supporting instructional strategies that are going to be used. In Table 4, attached to this document, describe many learning models according to the theories discussed. This table can be regarded as a starting guide, since it is necessary to consider that there might be other factors that can act upon the educational software development process, such as the peculiarities of the learner, his or her emotional structure, the main properties of the medium or technology to be used for the delivery of the instruction, and so on. Table 4 also relates educational theories with individual learning patterns contained in Table 1 to 3. In order to show that it is always possible to discover new patterns, two new ones are proposed and have been included in Table 4: **Mimic** (the learning activity is based on imitating behaviors or

procedures, strongly visual, a lot of sensorial information has to be processed) and **Expert Emulation** (Learners are placed in the role of experts so that they can acquire the expert's way of thinking and approaching problems, not the expert's knowledge). Many models may overlap, it is difficult to find an instructional environment that can fit entirely within a pure learning theory.

Based upon the requirements found for the educational software product and the instructional strategies recommended by different learning theories, the designer can develop a model for a new pattern that he will test in further steps of the design process.

Table 4. Learning Theories and Developmental Patterns

Learning Theory	Learning Environment main lineaments	Related Individual Ed. Soft. Development Patterns
Behaviorism Learning by imitation	• Traditional • Linear • Reactive • Process Training • Basic Skills • Passive Student • Instructive	• Tutorial • Coaching • Mimic
Cognitivism Learning by Association	• Complex • Goal Oriented • Decision Making • Intellectual Skills • Logic Reasoning • Interactive • Informative	• Simulation • Meandering • Current Information
Constructivism Learning by Experience	• Highly Complex • Ill-structured • Context dependent • Situated • Experimental • Users create artifacts • Interactive Cooperative	• Research/Construct • Scientific Method • Expert Emulation

6. Testing the right pattern: prototyping, evaluating and implementing

It is time for the designer to embed the set of instructional strategies in a first prototype of the product that can be submitted to the user for consideration and evaluation. This attitude on behalf of the designer is coherent with the user-centered approach recommended in section 3 of this paper. Upon receipt of the user's feedback, an iterative process of successive refinements and assessments is activated, until the final product meets all the requirements established by the user and satisfies him entirely. Fig. 1 depicts the entire development process.

If the designer wants to consider the risks involved before starting a new iteration of this cycle, he will be performing a developmental approach of type spiral as suggested by Barry Boehm in his risk-oriented software development paradigm [11].Once the product is ready to be implemented in the user's platform, new problems or bugs that might surface in this ultimate phase have to be properly fixed. If a new pattern for learning has been found, the designer's knowledge is enriched and enlarged [12].

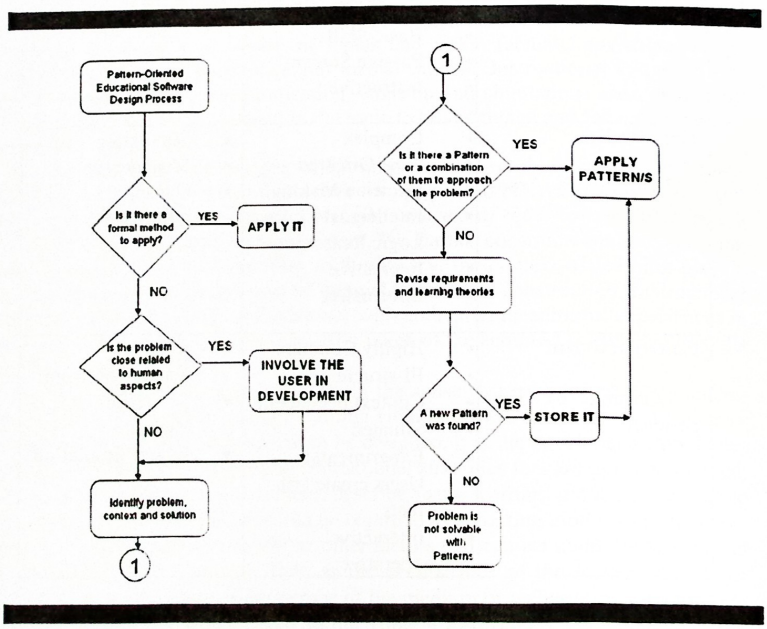

Fig. 1 A diagram showing the Pattern-Oriented Educational Software Development Process

7. Concluding Remarks

The Pattern-Oriented Paradigm is well suited for addressing the problem of educational software design, as can be seen in the relationships depicted by Table 4. Under this context, it is recommended to substitute traditional software development processes (which are normally applied to educational software products as well) by a process of finding the right pattern that best fits the design problem under consideration. This can be already known pattern or a new one, or any combination of both types. As a main advantage for this strategy arises the *reusability of ideas and experience*. Moreover, the reusability of ideas comprises generally the reusability of code, a dominant strategy provided by the Object-Oriented Paradigm of software design. That is why these two paradigms are closely related and even complementary of each other. AS a result of the Pattern-Oriented Paradigm, efforts, experiences, best practices are documented and it is possible to take advantage of and learn from them over and over again. As a disadvantage, the designer has to overcome the temptation of automatically applying already known patterns when certain superficial conditions hold, without previously carrying out a deep analysis of the problem and its requirements.

References

1. C. Alexander (1996). A Pattern Language. Oxford University Press.
2. R. Gabriel (1998). Patterns of Software: Tales from the software community. Oxford University Press.
3. H. Gardner (2006). Changing Minds: The art and science of changing our own and other people's minds. Boston MA. Harvard Business School Press.
4. J. Noyes et al. (1995). User involvement in the design process: a case for end-user evaluation of software packages. Human Centred Automation. IEE Colloquium, 27(3):2-3.
5. C. Alexander (1998) A Timeless Way of Building. Oxford University Press.
6. E. Sierra et al. (2001). Selección de Estrategias Instruccionales: Abordaje desde la Teoría del Conocimiento. Revista del Instituto Tecnológico de Buenos Aires. 26 (1): 39-57.
7. E. Sierra et al. (2003). Sistemas Expertos que recomiendan estrategias de instrucción: un modelo para su desarrollo. Revista Latinoamericana de Tecnología Educativa 1(1): 19-30.
8. M. Hannafin et al. (1998). The design, development and evaluation of instructional software. Prentice Hall.
9. R. Gagne et al (1992). Principles of Instructional Design.. Holt Rinehart & Winston.
10. J. Black et al (1992). Types of Knowledge Representation. CCTE Report.Teachers College. Columbia University Press.
11. B. Boehm (1981). Software Engineering Economics. Prentice Hall.
12. P. Avgeriu et al (2003). Towards a Pattern Language for Learning Management Systems. Journal of Educational Technology & Society. 6 (2): 11-24.